textsum谷歌最新地址： https://github.com/tensorflow/models/tree/master/research/textsum

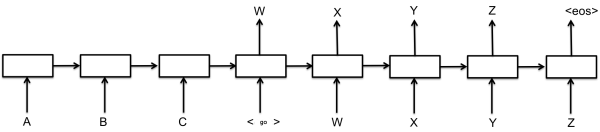
<https://github.com/zpppy/seq2seq-chinese-textsum>

**seq2seq-chinese-textsum**

近几十年来，获取新信息的方式发生了根本性变化，也带来了越来越多挑战。信息的获取已不再是瓶颈；瓶颈在于我们是否有能力紧跟信息的步伐。我们都必须通过越来越多的阅读来获取关于工作、新闻和社交媒体的最新进展。我们研究了人工智能在信息大潮中帮助人们提高工作能力的方法——答案之一是让算法自动归纳长文本。

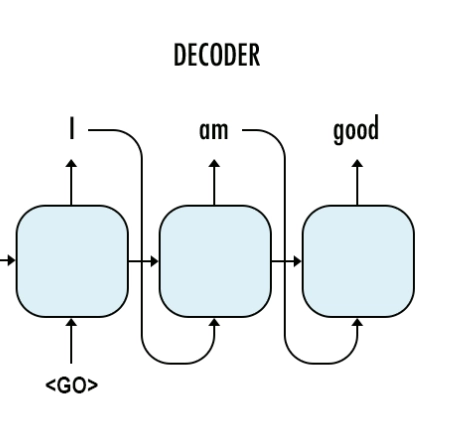
 本程序主要是利用seq2seq+Attention模型进行自动摘要生成。详细介绍请参[中文文本摘要生成](http://www.hackerfun.cn/article/2017/07/19/35/)

我们喂给它的训练集数据是由一系列{正文: 摘要}对组成. 假设正文是x=[x1,...,xM], M是正文词符的数量, 对应的摘要为y=[y1,...,yN], N是摘要单词的数量.

训练阶段输入   
encoder\_input  (正文编码)  
decoder\_input  (标题编码)  
decoder\_target  (标题编码shift一位)  
其中decoder\_target是decoder\_input的shift一位（如图）   


训练好模型之后，

预测阶段输入   
encoder\_input  (待预测的正文编码)  
decoder\_input：前一decoder的step的预测结果就会作为下一step的输入，只用输入一个<GO>（如图）

也就是说这边只用输入一个<GO>，就能自动预测下一个可能出现的单词，然后再预测下个可能出现的单词，直到指定长度为止  


（本例程中data下面的train里面的数据太少，训练出来效果不好，仅仅用作测试程序能通过。Dev里面是每个100个epoch进行一次测试的文件。）

使用方法：先运行headline.py训练模型，然后运行predict\_main.sh进行测试

Tensorflow 自动文摘: 基于Seq2Seq+Attention模型的Textsum模型

http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif(2017-11-09 11:43:02)

[[http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif](javascript:;)转载▼](javascript:;)

|  |  |
| --- | --- |
| 标签：  [it](http://search.sina.com.cn/?c=blog&q=it&by=tag)    [大数据](http://search.sina.com.cn/?c=blog&q=%B4%F3%CA%FD%BE%DD&by=tag)    [自动摘要](http://search.sina.com.cn/?c=blog&q=%D7%D4%B6%AF%D5%AA%D2%AA&by=tag) | 分类： [深度学习](http://blog.sina.com.cn/s/articlelist_6300143775_1_1.html) |

<https://github.com/rockingdingo/deepnlp/tree/master/deepnlp/textsum>

**简介**

这篇文章中我们将基于Tensorflow的Seq2Seq+Attention模型，介绍如何训练一个中文的自动生成新闻标题的模型。自动总结(Automatic Summarization)类型的模型一直是研究热点。 直接抽出重要的句子的抽取式方法较为简单，有如textrank之类的算法，而生成式(重新生成新句子)较为复杂，效果也不尽如人意。目前比较流行的Seq2Seq模型，由 Sutskever等人提出，基于一个Encoder-Decoder的结构将source句子先Encode成一个固定维度d的向量，然后通过Decoder部分一个字符一个字符生成Target句子。添加入了Attention注意力分配机制后，使得Decoder在生成新的Target Sequence时，能得到之前Encoder编码阶段每个字符的隐藏层的信息向量Hidden State，使得生成新序列的准确度提高。

**数据准备和预处理**

我们选择公开的“搜狐新闻数据(SogouCS)”的语料，包含2012年6月—7月期间的新闻数据，超过1M的语料数据，包含新闻标题和正文的信息。数据集可以从搜狗lab下载。 <http://www.sogou.com/labs/resource/cs.php>

数据的预处理阶段极为重要，因为在Encoder编码阶段处理那些信息，直接影响到整个模型的效果。我们主要对下列信息进行替换和处理：

* 特殊字符：去除特殊字符，如：“「，」,￥,…”；
* 括号内的内容：如表情符，【嘻嘻】，【哈哈】
* 日期：替换日期标签为TAG\_DATE，如：\*\*\*年\*月\*日，\*\*\*\*年\*月，等等
* 超链接URL：替换为标签TAG\_URL；
* 删除全角的英文：替换为标签TAG\_NAME\_EN；
* 替换数字：TAG\_NUMBER；

在对文本进行了预处理后，准备训练语料： 我们的Source序列，是新闻的正文，待预测的Target序列是新闻的标题。 我们截取正文的分词个数到MAX\_LENGTH\_ENC=120个词，是为了训练的效果正文部分不宜过长。标题部分截取到MIN\_LENGTH\_ENC = 30，即生成标题不超过30个词。

在data\_util.py类中，生成训练数据时做了下列事情：

* create\_vocabulary()方法创建词典；
* data\_to\_token\_ids()方法把训练数据(content-train.txt)转化为对应的词ID的表示；

**两个文件的格式:**

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 数据1 正文 content-train.txt
2. 世间 本 没有 歧视 TAG\_NAME\_EN 歧视 源自于 人 的 内心 活动 TAG\_NAME\_EN “ 以爱 之 名 ” TAG\_DATE 中国 艾滋病 反歧视 主题 创意 大赛 开幕 TAG\_NAME\_EN 让 “ 爱 ” 在 高校 流动 。 TAG\_NAME\_EN 详细 TAG\_NAME\_EN
3. 济慈 之 家 小朋友 感受 爱心 椅子  TAG\_DATE TAG\_NAME\_EN 思源 焦点 公益 基金 向 盲童 孤儿院 “ 济慈 之 家 ” 提供 了 首 笔 物资 捐赠 。 这 笔 价值 近 万 元 的 物资 为 曲 美 家具 向 思源 · 焦点 公益 基金 提供 的 儿童 休闲椅 TAG\_NAME\_EN 将 用于 济慈 之 家 的 小孩子们 日常 使用 。
4. ...

**[python]**[**view plain**](http://blog.csdn.net/rockingdingo/article/details/55224282)[**copy**](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 数据2 标题 title-train.txt
2. 艾滋病 反歧视 创意 大赛
3. 思源 焦点 公益 基金 联手 曲 美 家具 共 献 爱心
4. ...

**训练模型**

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. #代码1
2. python headline.py

**预测**

**运行predict.py, 交互地输入分好词的文本, 得到textsum的结果**

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. #代码2-1
2. python predict.py
4. # 输入和输出
5. #> 中央 气象台 TAG\_DATE TAG\_NUMBER 时 继续 发布 暴雨 蓝色 预警 TAG\_NAME\_EN 预计 TAG\_DATE TAG\_NUMBER 时至 TAG\_DATE TAG\_NUMBER 时 TAG\_NAME\_EN 内蒙古 东北部 、 山西 中 北部 、 河北 中部 和 东北部 、 京津 地区 、 辽宁 西南部 、 吉林 中部 、 黑龙江 中部 偏南 等 地 的 部分 地区 有 大雨 或 暴雨 。
6. #current bucket id0
7. #中央 气象台 发布 暴雨 蓝色 预警
8. #>

我们尝试输入下列分好词的新闻正文，一些挑选过的自动生成的中文标题如下:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 新闻正文 | 新闻标题 | textsum自动生成标题 |
| 469 | 中央 气象台 TAG\_DATE TAG\_NUMBER 时 继续 发布 暴雨 蓝色 预警 TAG\_NAME\_EN 预计 TAG\_DATE TAG\_NUMBER 时至 TAG\_DATE TAG\_NUMBER 时 TAG\_NAME\_EN 内蒙古 东北部 、 山西 中 北部 、 河北 中部 和 东北部 、 京津 地区 、 辽宁 西南部 、 吉林 中部 、 黑龙江 中部 偏南 等 地 的 部分 地区 有 大雨 或 暴雨 。 | 中央 气象台 继续 发布 暴雨 预警 北京 等 地 有 大雨 | 中央 气象台 发布 暴雨 蓝色 预警 |
| 552 | 美国 科罗拉多州 山林 大火 持续 肆虐 TAG\_NAME\_EN 当地 时间 TAG\_DATE 横扫 州 内 第二 大 城市 科罗拉多斯 普林斯 一 处 居民区 TAG\_NAME\_EN 迫使 超过 TAG\_NUMBER TAG\_NAME\_EN TAG\_NUMBER 万 人 紧急 撤离 。 美国 正 值 山火 多发 季 TAG\_NAME\_EN 现有 TAG\_NUMBER 场 山火 处于 活跃 状态 。 | 山火 横扫 美 西部 TAG\_NUMBER 州 奥 巴马 将 赴 灾区 视察 联邦 调查局 介入 查 原因 | 美国 科罗拉多州 山火 致 TAG\_NUMBER 人 死亡 |
| 917 | 埃及 选举 委员会 昨天 宣布 TAG\_NAME\_EN 穆斯林 兄弟会 下属 自由 与 正义党 主席 穆尔西 获得 TAG\_NUMBER TAG\_NAME\_EN TAG\_NUMBER TAG\_NAME\_EN 的 选票 TAG\_NAME\_EN 以 微弱 优势 击败 前 总理 沙 菲克 赢得 选举 TAG\_NAME\_EN 成为 新任 埃及 总统 。 媒体 称 其 理念 获 下层 民众 支持 。 | 埃及 大选 昨晚 结束 新 总统 穆尔西 被 认为 具有 改革 魄力 | 埃及 总统 选举 结果 |
| 920 | 上 周 TAG\_NAME\_EN 广东 华兴 银行 在 央行 宣布 降息 和 调整 存贷款 波幅 的 第二 天 TAG\_NAME\_EN 立即 宣布 首 套 房贷 利率 最低 执行 七 折 优惠 。 一 石 激起 千层 浪 TAG\_NAME\_EN 随之 而 起 的 “ 房贷 七 折 利率 重 出 江湖 ” 和 “ 房地产 调控 松绑 ” 的 谣言 四起 。 | 房贷 “ 七 折 利率 ” 真相 调查 TAG\_NAME\_EN 符合 条件 的 几乎 为零 | 银监会 否认 房贷 房贷 利率 |

**预测并计算ROUGE评估**

运行predict.py, 同时调用eval.py 中的 evaluate(X, Y, method = "rouge\_n", n = 2) 方法计算ROUGE分

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. #代码2-2 linux shell
2. folder\_path=`pwd`
3. input\_dir=${folder\_path}/news/test/content-test.txt
4. reference\_dir=${folder\_path}/news/test/title-test.txt
5. summary\_dir=${folder\_path}/news/test/summary.txt
7. python predict.py $input\_dir $reference\_dir $summary\_dir
9. # 输出:
10. # 中央 气象台 发布 暴雨 蓝色 预警
11. # Evaludated Rouge-2 score is 0.1818
12. # ...

下面我们将具体介绍tensorflow的seq2seq模型如何实现，首先先简单回顾模型的结构。

**Seq2Seq+Attention模型回顾**

Seq2Seq模型有效地建模了基于输入序列，预测未知输出序列的问题。模型有两部分构成，一个编码阶段的”Encoder”和一个解码阶段的”Decoder”。如下图的简单结构所示，Encoder的RNN每次输入一个字符代表的embedding向量，如依次输入A,B,C, 及终止标志，将输入序列编码成一个固定长度的向量；之后解码阶段的RNN会一个一个字符地解码, 如预测为X, 之后在训练阶段会强制将前一步解码的输出作为下一步解码的输入，如X会作为下一步预测Y时的输入。

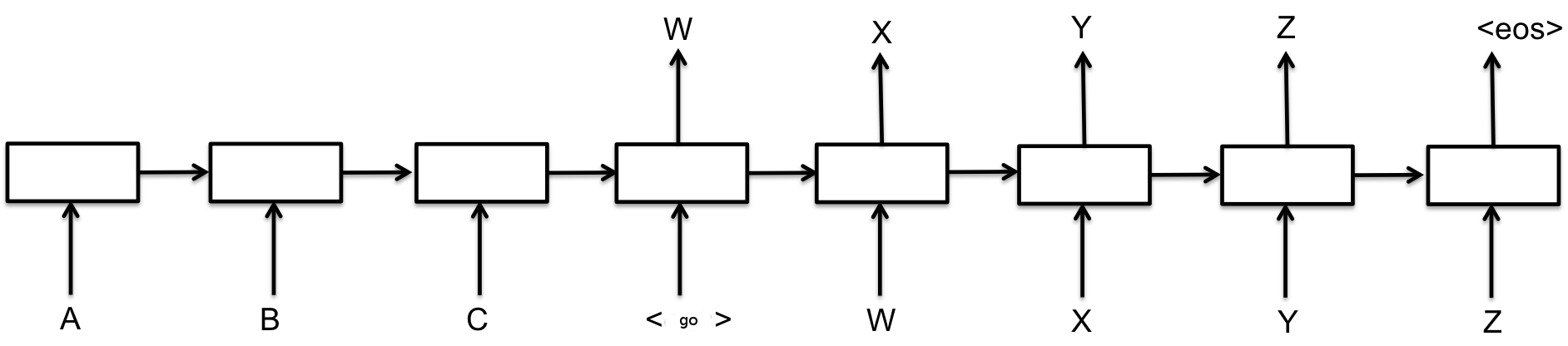


图1 Seq2Seq model

定义输入序列  ，由Tx个固定长度为d的向量构成； 输出序列为  ，由Ty个固定长度为d的向量构成； 定义输入序Encoder阶段的RNN隐藏层为 hj, Decoder阶段的RNN隐藏层为 Si

**Attention注意力分配机制**

LSTM模型虽然具有记忆性，但是当Encoder阶段输入序列过长时，解码阶段的LSTM也无法很好地针对最早的输入序列解码。基于此，Attention注意力分配的机制被提出，就是为了解决这个问题。在Decoder阶段每一步解码，都能够有一个输入，对输入序列所有隐藏层的信息h\_1,h\_2,…h\_Tx进行加权求和。打个比方就是每次在预测下一个词时都会把所有输入序列的隐藏层信息都看一遍，决定预测当前词时和输入序列的那些词最相关。

Attention机制代表了在解码Decoder阶段，每次都会输入一个Context上下文的向量Ci, 隐藏层的新状态Si根据上一步的状态Si-1, Yi, Ci 三者的一个非线性函数得出。

Context向量在解码的每一步都会重新计算，根据一个MLP模型计算出输出序列i对每个输入序列j的隐含层的对应权重aij，并对所有隐含层加权平均。文章中说的Alignment Model就是代表这种把输入序列位置j和输出序列位置i建立关系的模型。

aij 即可以理解为Decoder解码输出序列的第i步，对输入序列第j步分配的注意力权重。

eij为一个简单的MLP模型激活的输出；aij的计算是对eij做softmax归一化后的结果。

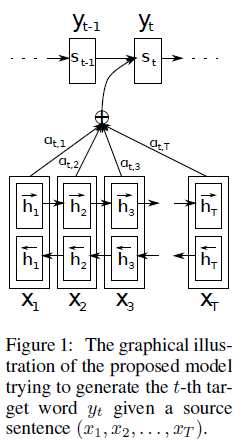


图2 Translate Seq2Seq alignment model

**Soft Attention和Hard Attention区别**

Soft Attention通常是指以上我们描述的这种全连接(如MLP计算Attention 权重)，对每一层都可以计算梯度和后向传播的模型；不同于Soft attention那样每一步都对输入序列的所有隐藏层hj(j=1….Tx) 计算权重再加权平均的方法，Hard Attention是一种随机过程，每次以一定概率抽样，以一定概率选择某一个隐藏层 hj\*，在估计梯度时也采用蒙特卡罗抽样Monte Carlo sampling的方法。

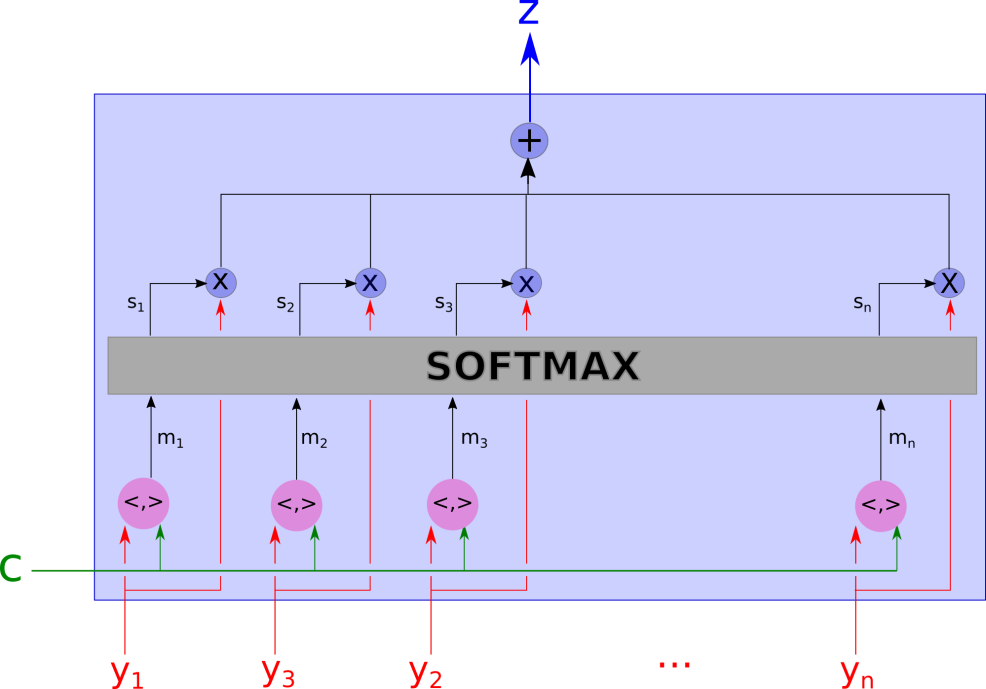


图3 Soft Attention 模型

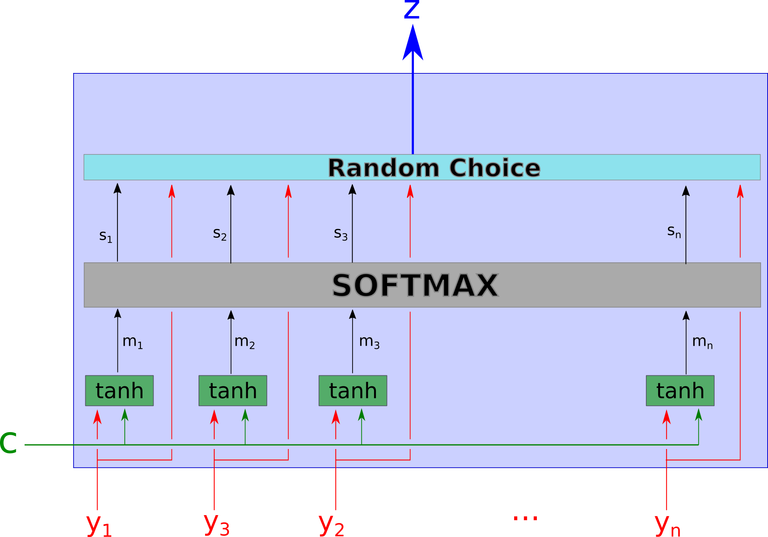


图4 Hard Attention 模型

**模型实现**

我们对Tensorflow基本教程里的translate英语法语翻译例子里的seq2seq\_model.py类稍加修改，就能够符合我们textsum例子使用，另外我们还会分析针对英文的textsum教程中构建双向Bi-LSTM的Encoder-Decoder的例子。

1.Seq2Seq模型文件: seq2Seq\_model.py

**单向LSTM的Encoder-Decoder结构**

教程中的例子很长，但是将实例代码分解来看不是那么复杂，下面将分三段来介绍官方tutorial里的如何构建seq2seq模型。

定义基本单元: 多层LSTM cell

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. #代码3-1
2. # Create the internal multi-layer cell for our RNN.
3. single\_cell = tf.nn.rnn\_cell.GRUCell(size) # default use GRU
4. **if** use\_lstm:
5. single\_cell = tf.nn.rnn\_cell.BasicLSTMCell(size, state\_is\_tuple=True)
6. cell = single\_cell
7. **if** num\_layers > 1:
8. cell = tf.nn.rnn\_cell.MultiRNNCell([single\_cell] \* num\_layers, state\_is\_tuple=True)

定义前向过程的 seq2seq\_f 函数，利用tf.nn.seq2seq.embedding\_attention\_seq2seq 返回output

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 代码3-2
2. # The seq2seq function: we use embedding for the input and attention.
3. **def** seq2seq\_f(encoder\_inputs, decoder\_inputs, do\_decode):
4. **return** tf.nn.seq2seq.embedding\_attention\_seq2seq(
5. encoder\_inputs,
6. decoder\_inputs,
7. cell,
8. num\_encoder\_symbols=source\_vocab\_size,
9. num\_decoder\_symbols=target\_vocab\_size,
10. embedding\_size=size,
11. output\_projection=output\_projection,
12. feed\_previous=do\_decode,
13. dtype=tf.float32)

桶Bucket的机制: 应用Bucket机制，核心的思想是把输入序列的句子按照长度的相似程度分到不同的固定长度的Bucket里面，长度不够的都添加PAD字符。之所以有Bucket的原因是工程效率：”RNN在数学上是可以处理任意长度的数据的。我们在TensorFlow中使用bucket的原因主要是为了工程实现的效率” （摘自知乎JQY的回答https://www.zhihu.com/question/42057513）

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 代码3-3
2. # Training outputs and losses.
3. **if** forward\_only:
4. self.outputs, self.losses = tf.nn.seq2seq.model\_with\_buckets(
5. self.encoder\_inputs, self.decoder\_inputs, targets,
6. self.target\_weights, buckets, **lambda** x, y: seq2seq\_f(x, y, True),
7. softmax\_loss\_function=softmax\_loss\_function)
8. # If we use output projection, we need to project outputs for decoding.
9. **if** output\_projection **is** **not** None:
10. **for** b **in** xrange(len(buckets)):
11. self.outputs[b] = [
12. tf.matmul(output, output\_projection[0]) + output\_projection[1]
13. **for** output **in** self.outputs[b]
14. ]
15. **else**:
16. self.outputs, self.losses = tf.nn.seq2seq.model\_with\_buckets(
17. self.encoder\_inputs, self.decoder\_inputs, targets,
18. self.target\_weights, buckets, **lambda** x, y: seq2seq\_f(x, y, False),
19. softmax\_loss\_function=softmax\_loss\_function)

tf.nn.seq2seq.model\_with\_buckets()结果返回output序列；’ buckets = [(120, 30), (200, 35), (300, 40), (400, 40), (500, 40)]

https://github.com/tensorflow/tensorflow/blob/64edd34ce69b4a8033af5d217cb8894105297d8a/tensorflow/contrib/legacy\_seq2seq/python/ops/seq2seq.py

**双向LSTM的Encoder-Decoder结构**

Encoder阶段自己定义了Bi-LSTM结构

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 代码4-1
2. # URL: https://github.com/tensorflow/models/tree/master/textsum
3. # Encoder: Multi-Layer LSTM, Output: encoder\_outputs
4. **for** layer\_i **in** xrange(hps.enc\_layers):
5. with tf.variable\_scope('encoder%d'%layer\_i), tf.device(
6. self.\_next\_device()):
7. cell\_fw = tf.nn.rnn\_cell.LSTMCell(
8. hps.num\_hidden,
9. initializer=tf.random\_uniform\_initializer(-0.1, 0.1, seed=123),
10. state\_is\_tuple=False)
11. cell\_bw = tf.nn.rnn\_cell.LSTMCell(
12. hps.num\_hidden,
13. initializer=tf.random\_uniform\_initializer(-0.1, 0.1, seed=113),
14. state\_is\_tuple=False)
15. (emb\_encoder\_inputs, fw\_state, \_) = tf.nn.bidirectional\_rnn(
16. cell\_fw, cell\_bw, emb\_encoder\_inputs, dtype=tf.float32,
17. sequence\_length=article\_lens)
18. encoder\_outputs = emb\_encoder\_inputs
20. with tf.variable\_scope('output\_projection'):
21. w = tf.get\_variable(
22. 'w', [hps.num\_hidden, vsize], dtype=tf.float32,
23. initializer=tf.truncated\_normal\_initializer(stddev=1e-4))
24. w\_t = tf.transpose(w)
25. v = tf.get\_variable(
26. 'v', [vsize], dtype=tf.float32,
27. initializer=tf.truncated\_normal\_initializer(stddev=1e-4))
28. Decoder阶段，利用内置的tf.nn.seq2seq.attention\_decoder()函数返回output

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 代码4-2
2. with tf.variable\_scope('decoder'), tf.device(self.\_next\_device()):
3. # When decoding, use model output from the previous step
4. # for the next step.
5. loop\_function = None
6. **if** hps.mode == 'decode':
7. loop\_function = \_extract\_argmax\_and\_embed(
8. embedding, (w, v), update\_embedding=False)
10. cell = tf.nn.rnn\_cell.LSTMCell(
11. hps.num\_hidden,
12. initializer=tf.random\_uniform\_initializer(-0.1, 0.1, seed=113),
13. state\_is\_tuple=False)
15. encoder\_outputs = [tf.reshape(x, [hps.batch\_size, 1, 2\*hps.num\_hidden])
16. **for** x **in** encoder\_outputs]
17. self.\_enc\_top\_states = tf.concat(1, encoder\_outputs)
18. self.\_dec\_in\_state = fw\_state
19. # During decoding, follow up \_dec\_in\_state are fed from beam\_search.
20. # dec\_out\_state are stored by beam\_search for next step feeding.
21. initial\_state\_attention = (hps.mode == 'decode')
22. decoder\_outputs, self.\_dec\_out\_state = tf.nn.seq2seq.attention\_decoder(
23. emb\_decoder\_inputs, self.\_dec\_in\_state, self.\_enc\_top\_states,
24. cell, num\_heads=1, loop\_function=loop\_function,
25. initial\_state\_attention=initial\_state\_attention)

**Attention注意力矩阵的可视化**

Attention 注意力分配机制的权重矩阵[Aij]可以反映在Decoder阶段第i个输出字符对Encoder阶段的第j个字符的注意力分配的权重aij。 我们可以通过绘制Heatmap来可视化seq2seq模型中Decoder的Y对Encoder的X每个字符的权重。

**获取attention\_mask的值**

attention\_mask 即为我们感兴趣的注意力权重分配的tensor，我们首先来看tensorflow的源码seq2seq.py这个ops的实现， 容易发现，计算attention\_mask 变量a的代码出现在 attention\_decoder()函数内的attention()函数体下, a = nn\_ops.softmax(s) 这句。 我们把该变量添加到return语句中的返回值，同时也修改所有调用了attention\_decoder()的上层的函数，为了最终能够在主函数中将attn\_mask这个变量抽取出。 具体需要修改的脚本参考textsum项目下的seq2seq\_attn.py这个文件。 之后我们在主函数中利用attn\_out = session.run(self.attn\_masks[bucket\_id], input\_feed) ，对变量进行session.run() 就可以获得当前这个样本的attention矩阵的值。

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 代码5-1
2. # URL: https://github.com/rockingdingo/deepnlp/blob/master/deepnlp/textsum/seq2seq\_attn.py
4. **def** attention\_decoder():
5. ## some code
7. **def** attention(query):
8. """Put attention masks on hidden using hidden\_features and query."""
9. ds = []
10. **if** nest.is\_sequence(query):
11. query\_list = nest.flatten(query)
12. **for** q **in** query\_list:
13. ndims = q.get\_shape().ndims
14. **if** ndims:
15. **assert** ndims == 2
16. query = array\_ops.concat\_v2(query\_list, 1)
17. **for** a **in** xrange(num\_heads):
18. with variable\_scope.variable\_scope("Attention\_%d" % a):
19. y = linear(query, attention\_vec\_size, True)
20. y = array\_ops.reshape(y, [-1, 1, 1, attention\_vec\_size])
21. # Attention mask is a softmax of v^T \* tanh(...).
22. s = math\_ops.reduce\_sum(v[a] \* math\_ops.tanh(hidden\_features[a] + y),
23. [2, 3])
25. # Tensor a 即为我们需要抽取的attention\_mask
26. a = nn\_ops.softmax(s)
28. d = math\_ops.reduce\_sum(
29. array\_ops.reshape(a, [-1, attn\_length, 1, 1]) \* hidden, [1, 2])
30. ds.append(array\_ops.reshape(d, [-1, attn\_size]))
31. **return** ds, a

**利用matplotlib可视化**

我们利用matplotlib包中绘制heatmap的函数，可以简单地将上一步抽取出的attn\_matrix可视化。在eval.py模块中我们整合了一个eval.plot\_attention(data, X\_label=None, Y\_label=None) 函数来简单绘制attention权重矩阵。 运行 predict\_attn.py 脚本，输入分好词的待分析新闻文本，然后自动生成的jpg图片就保存在./img目录下。

**[python]** [view plain](http://blog.csdn.net/rockingdingo/article/details/55224282) [copy](http://blog.csdn.net/rockingdingo/article/details/55224282)

1. # 代码5-2
2. # 输入文本, 查看Attention的heatmap:
3. # > 中央 气象台 TAG\_DATE TAG\_NUMBER 时 继续 发布 暴雨 蓝色 预警 TAG\_NAME\_EN 预计 TAG\_DATE TAG\_NUMBER 时至 TAG\_DATE TAG\_NUMBER 时 TAG\_NAME\_EN 内蒙古 东北部 、 山西 中 北部 、 河北 中部 和 东北部 、 京津 地区 、 辽宁 西南部 、 吉林 中部 、 黑龙江 中部 偏南 等 地 的 部分 地区 有 大雨 或 暴雨 。
5. python predict\_attn.py

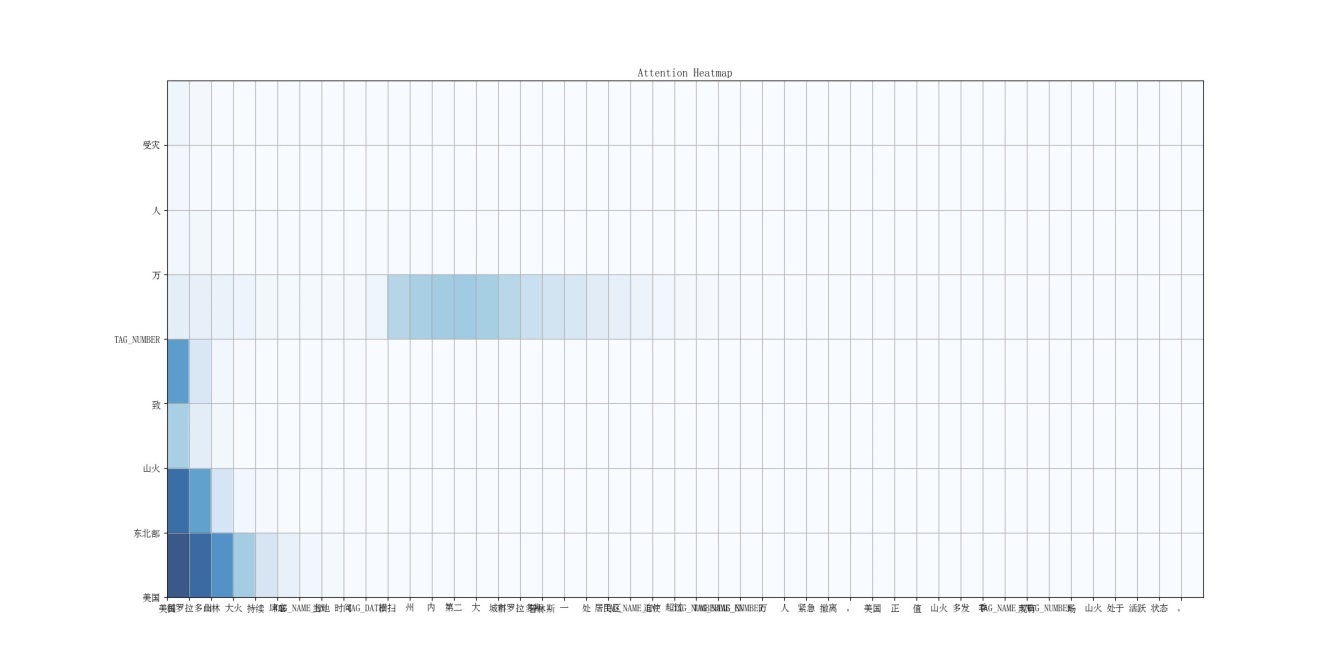


图5 Attention注意力权重分配的Heatmap

可视化部分代码参考项目中的 [predict\_attn.py](https://github.com/rockingdingo/deepnlp/tree/master/deepnlp/textsum/predict_attn.py) [seq2seq\_attn.py](https://github.com/rockingdingo/deepnlp/blob/master/deepnlp/textsum/seq2seq_attn.py) [seq2seq\_model\_attn.py](https://github.com/rockingdingo/deepnlp/tree/master/deepnlp/textsum/seq2seq_model_attn.py) 这三个文件。

**预测阶段Decode策略: 贪心算法和Beam Search**

贪心算法 Beam Search

**拓展阅读**

* [d](https://github.com/rockingdingo/deepnlp/tree/master/deepnlp/textsum)eepnlp Python包
* <http://www.deepnlp.org/>
* [Github源码textsum](https://github.com/rockingdingo/deepnlp/tree/master/deepnlp/textsum)
* <https://github.com/rockingdingo/deepnlp/tree/master/deepnlp/textsum>
* [Tensorflow seq2seq.py](https://github.com/tensorflow/tensorflow/blob/64edd34ce69b4a8033af5d217cb8894105297d8a/tensorflow/contrib/legacy_seq2seq/python/ops/seq2seq.py)
* <https://github.com/tensorflow/tensorflow/blob/64edd34ce69b4a8033af5d217cb8894105297d8a/tensorflow/contrib/legacy_seq2seq/python/ops/seq2seq.py>
* [Tensorflow Bi-LSTM textsum examples](https://github.com/tensorflow/models/tree/master/textsum)
* <https://github.com/tensorflow/models/tree/master/textsum>
* [ATTENTION MECHANISM](https://blog.heuritech.com/2016/01/20/attention-mechanism/)
* <https://blog.heuritech.com/2016/01/20/attention-mechanism/>

<http://blog.sina.com.cn/s/blog_17784909f0102xyak.html>

**TensorFlow文本摘要生成 - 基于注意力的序列到序列模型**

更新时间：2017-04-06 11:26:59点击次数：1469次

1 相关背景

维基百科对自动摘要生成的定义是, “使用计算机程序对一段文本进行处理, 生成一段长度被压缩的摘要, 并且这个摘要能保留原始文本的大部分重要信息”. 摘要生成**算法**主要分为抽取型(Extraction-based)和概括型(Abstraction-based)两类. 传统的摘要生成系统大部分都是抽取型的, 这类方法从给定的文章中, 抽取关键的句子或者短语, 并重新拼接成一小段摘要, 而不对原本的内容做创造性的修改. 这类抽取型算法工程上已经有很多开源的解决办法了, 例如Github上的项目sumy, pytextrank, textteaser等. 本文重点讲概括型摘要生成系统的算法思想和tensorflow实战, 算法思想源于A Neural Attention Model for Abstractive Sentence Summarization这篇论文. 本文希望帮助读者详细的解析算法的原理, 再结合github上相关的开源项目textsum讲解工程上的实际应用.本文由PPmoney**大数据**算法团队撰写，PPmoney是国内领先的互联网金融公司，旗下PPmoney理财总交易额超过700亿元。此外，若对TensorFlow的使用技巧和方法感兴趣，欢迎阅读本团队负责人黄文坚所著的《TensorFlow实战》。

2 算法原理

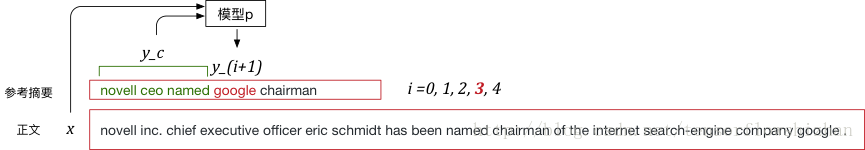
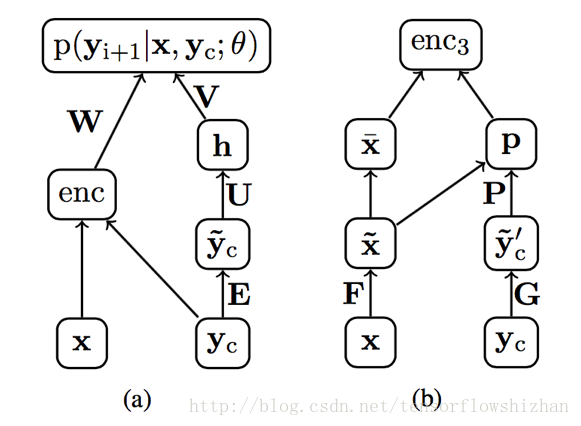
下面对A Neural Attention Model for Abstractive Sentence Summarization这篇文章, 的算法原理进行讲解. 我们将这个模型简称为NAM. 主要分为模型训练(train)和生成摘要(decode)两部分讲解.

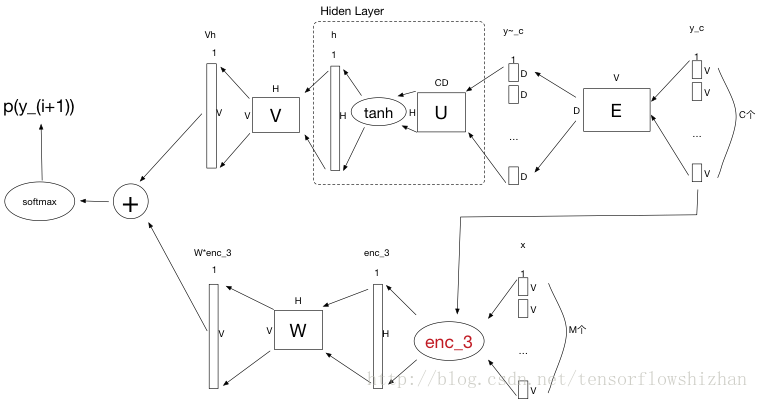
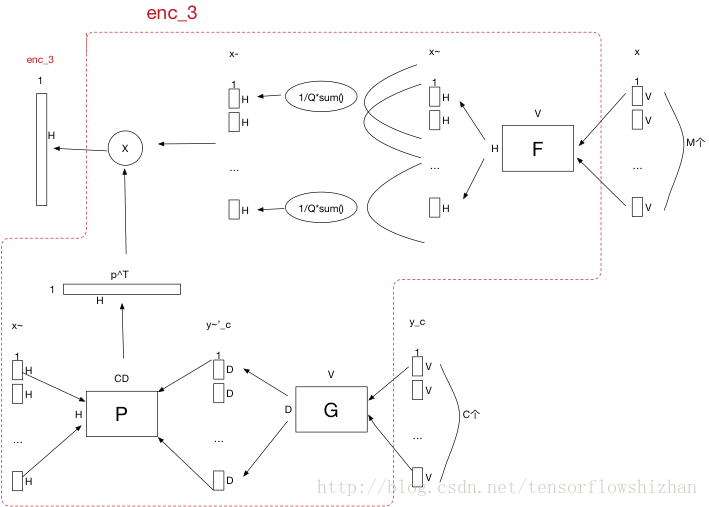
2.1 模型训练(train)

NAM这个模型是纯数据驱动, 我们喂给它的训练集数据是由一系列{正文: 摘要}对组成. 假设正文是x=[x1,...,xM], M是正文词符的数量, 对应的摘要为y=[y1,...,yN], N是摘要单词的数量.   
对于给定的数据, 我们希望给定x生成摘要为y的概率最大, 即maxθlogp(y|x;θ), θ是模型的参数. 但这个很难求解, 实际中我们用序列化的方式实例化这个目标, 原来的目标函数变为: 

maxθ∑i=0N−1logp(yi+1|x,yc;θ)

这里 yi+1是要预测的下一个词, yc≜y[i−C+1,...,i]是已知的序列, C是已知序列窗口的长度. 后面会提到, 这个窗口的位置也是注意力关注的位置, 在后面的训练过程中会根据学习到的权重调整不同位置注意力的概率大小. 这个窗口是随着i的迭代来滑动的.   
**参数说明**:   
y: 参考摘要所有单词向量组成的序列   
x: 正文的所以单词向量组成的序列   
i: 当前评估函数所对应的位置   
yc: 当前训练的窗口对应的局部摘要序列   
yi+1: 模型要预测的下一个单词

下面我们举一个例子来说明训练的过程:   
   
我们希望根据, 当前局部摘要序列yc和全部的正文信息x, 来预测下一个单词yi+1. 我们希望模型预测下一个单词为yi+1的概率最大, 并且希望所有单词都尽可能的预测准确, 在公式上表现为∑N−1i=0logp(yi+1|x,yc;θ)最大. 窗口C会从摘要的起始位置滑动到终止位置, 当i<C时, yc超出摘要的部分用起始符号<s>来补全.   
我们感兴趣的分布p(yi+1|x,yc;θ)是基于输入语句x的条件语言模型. 这里我们直接将原始的分布, 参数化为一个神经网络. 这个神经网络既包括了一个神经概率语言模型(neural probabilistic language model), 也包括了一个编码器(这个编码器就是一个条件摘要模型).   
通过包含编码器并且联合训练这两个组块, 我们根据当前yc对x的不同内容投入不同的关注度, 进而的到更好的结果. 模型结构如下图所示:   


* 模型整体的网络结构图(具有一个额外的编码器单元):   
  **右侧分支**: 仅根据当前的序列yc预测下一个单词是yi+1的概率, E是词嵌入, ỹ ′c -> h包括加权和激活函数的操作.   
  **左侧分支**: 使用yc和x生成隐层的下一个输出, yc会对encoder产生影响, 让encoder更多的关注x中与yc有关的内容.   
  **联合输出**: 最终结合右侧的神经语言模型和左侧attention-based编码器的输出, 求下一个词是yi+1的概率.
* 基于注意力模型的编码器enc31的网络结构图:   
  **左侧分支**: F是词嵌入矩阵, x̃  -> x¯是做了一下平滑处理.   
  **右侧分支**: G是词嵌入矩阵, 根据当前的y′c, 对x̃ 的不同位置投入不同的注意力, 并形成一个加权向量.   
  **联合输出**: 此时p已经携带了注意力的信息, 用p对平滑后的x¯再做加权, 得到encoder的输出.   
  下面两幅图分别是对整体结构和编码器结构的展开:   
     
  

**感兴趣的同学可以结合原文中的公式理解:**   
上图(a)中对应的公式: 

p(yi+1|x,yc;θ)∝exp(Vh+Wenc(x,yc)),yc~=[Eyi−C+1,...,Eyi],h=tanh(Uyc~)

参数是:   
θ=(E,U,V,W),   
E∈ℝD×V, 是一个词嵌入矩阵;   
U∈ℝ(CD)×H,V∈ℝV×H,W∈ℝV×H, 是权重矩阵.   
上图(b)中对应的公式:

enc3(x,yc)=pTx¯,p∝exp(x̃ Pyc~′),x̃ =[Fx1,...,FxM],yc~′=[Gyi−C+1,...,Gyi],∀i,x¯i=∑q=i−Qi+Qx̃ i/Q

这里G∈ℝD×V是一个内容的嵌入, P∈ℝH×(CD)是一个新的权重矩阵参数, Q是一个平滑窗口.   
**Mini-batch训练**   
这个模型是纯数据驱动的, 只要给它{正文: 摘要}训练集就能完成训练. 一旦我们已经定义了局部条件模型p(yi+1|x,yc;θ), 我们就能估计参数来最小化摘要集合的负对数似然函数. 假设训练集由J个输入-摘要对组成(x(1),y(1)),...,(x(J),y(J)). 负对数似然函数作用到摘要的每一个词, 即 

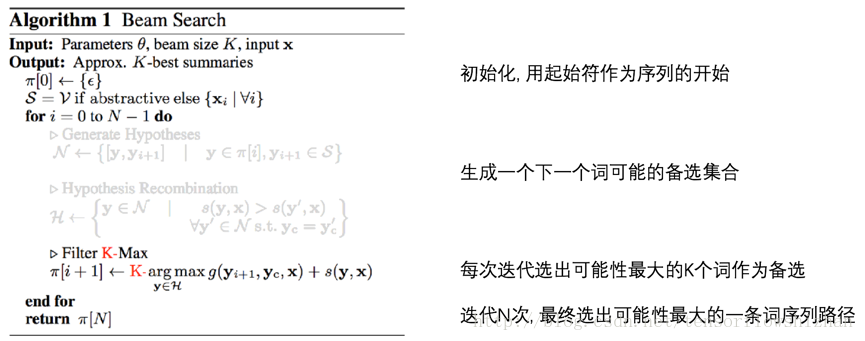
NLL(θ)=−∑j=1Jlogp(y(j)|x(j);θ)=−∑j=1J∑i=1N−1logp(y(j)i+1|x(j),yc;θ)

我们通过使用mini-batch和随机梯度下降最小化NLL.

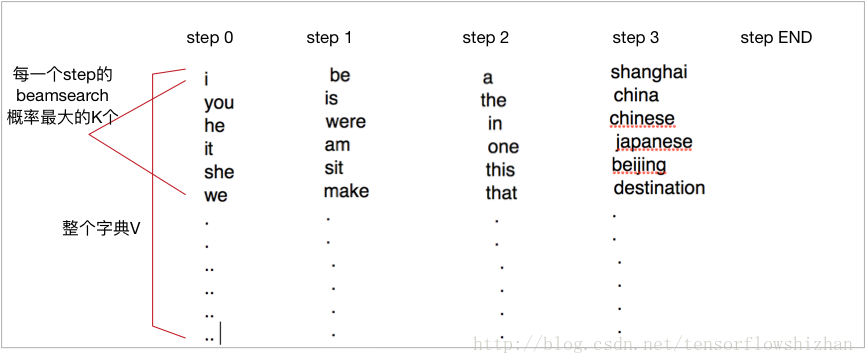
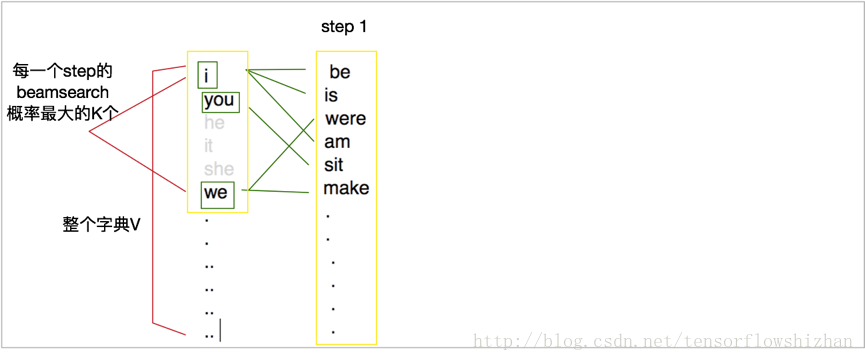
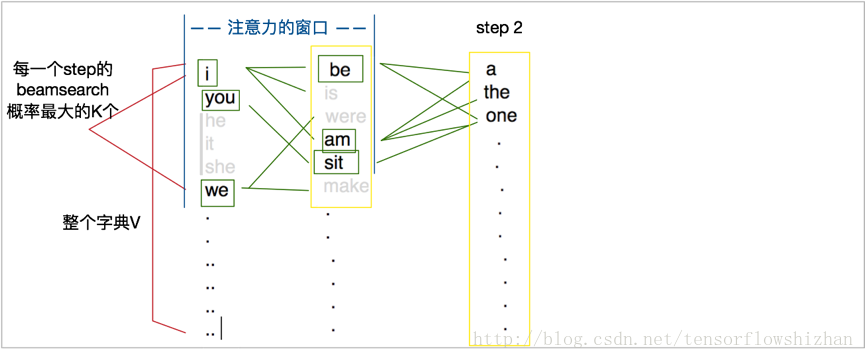
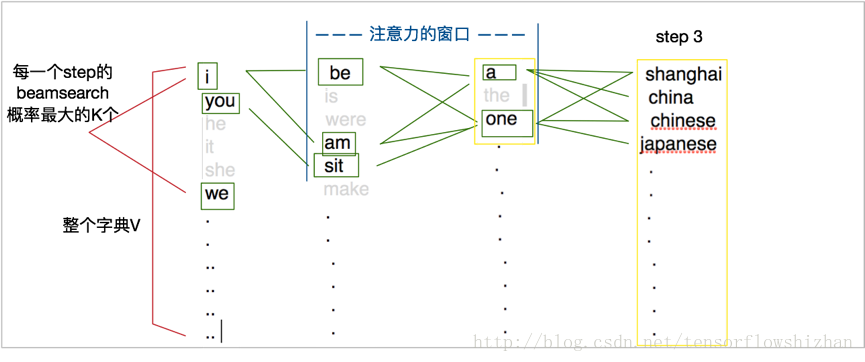
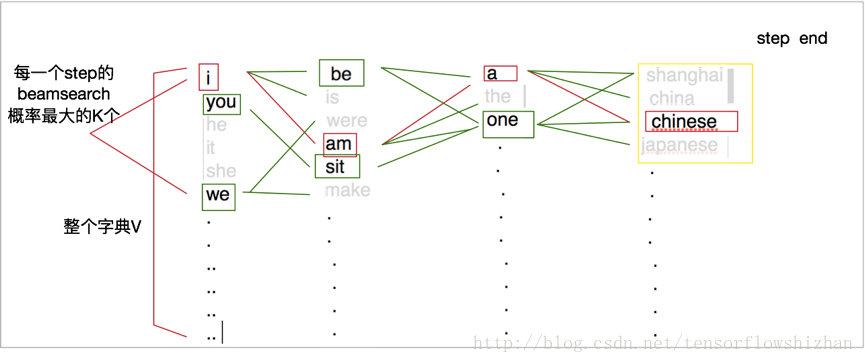
2.2 Beam Search生成摘要(decode)

我们现在回到生成摘要的问题. 回顾前面, 我们的目标是找到: 

y∗=argmaxy∈∑i=0N−1logp(yi+1|x,yc;θ)

是长度为N的序列y组成的集合, 如果字典中的单词数量是V的话, 我们要生成的这个摘要就有VN种可能性. 因为我们这里已经做了处理, 只根据前面的C个已经预测出的单词yc来预测下一个词yi+1. 这样算法复杂度变成了O(NVC). 但是即使是这样, 这个算法也太复杂了.   
使用维特比译码需要O(NVC).复杂度获得精确的解. 然而在实际中V太大使得问题难解. 一个替代方法是使用贪婪解来近似获得argmax, 只保证每次前进的一小步是概率最大的.   
在精确解和贪婪解方法之间取一个折中, 就是beam-search束搜索解码器(Algorithm1), 它在保持全量字典V的同时, 在输出摘要的每一个位置上将自己限制在K个潜在的假设内. 这种beam-search方法在神经机器翻译模型NMT也很常用. Beam search算法展示如下:   
   
参数说明:   
N: 摘要的长度   
K: beam的尺寸   
V: 字典里所有单词的数量   
C: 关注的词序列的长度

Beam search案例

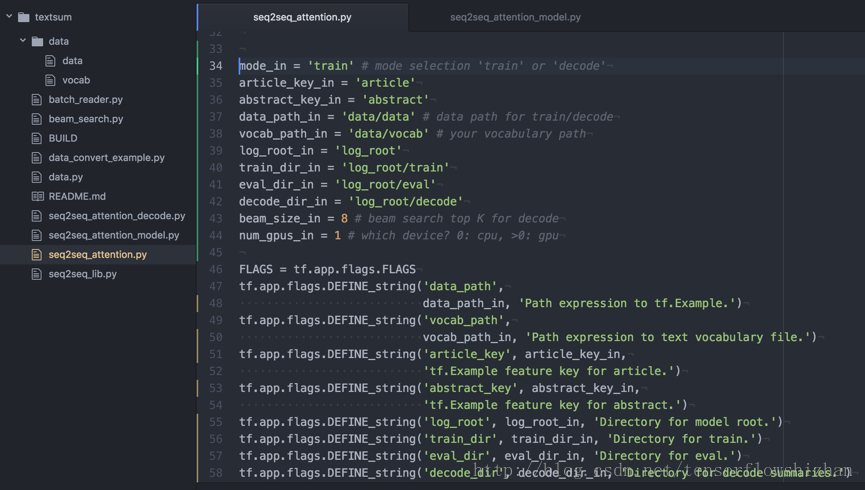
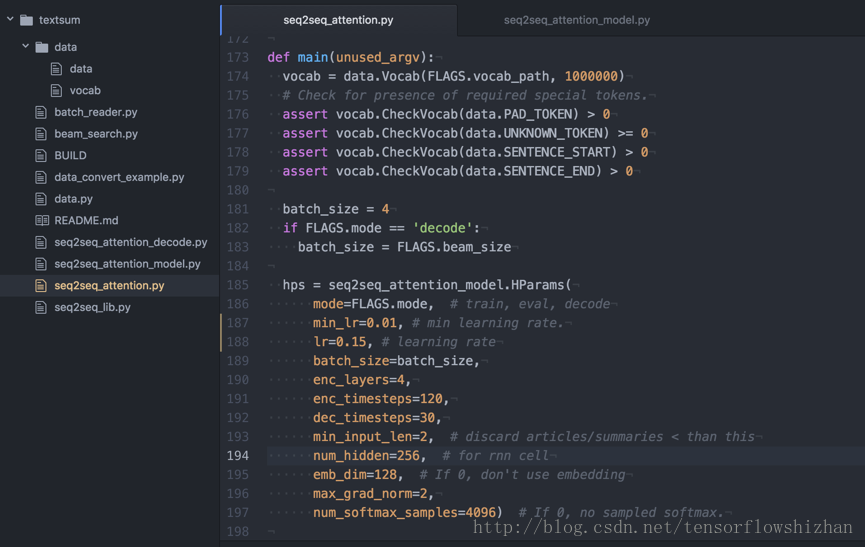
下面举一个简单的例子来说明beam search算法的运行过程. 在这个例子里, 摘要长度N=4, beam的大小K=6, 注意力窗口大小C=2, 模型最理想的结果是‘i am a chinese’. Beamsearch的每一次迭代都从字典V里找K个最大的可能.   
   
Step1: 预测前C个词的时候窗口溢出的部分需要进行padding操作, 预测第1个词的时候我们选出K个词符.   
   
Step2: 预测第2个词的时候, 我们选出新的K个词符, 对应K条备选路径. 前一阶段概率低的路径和词符, 被抛弃掉.   
   
Step3: 重复前面的过程.   
   
Step4: 每次beam search不一定能选出不同的K个词, 但是每次beam search都找到最优的前K个路径, 路径可以有重叠.   
   
Step5: 迭代N次, 最终选出可能性最大的一条词序列路径   
   
下面是对Beam Search算法的详细分析, 对原文的Algorithm 1逐条进行解释.

Beam Search算法分析

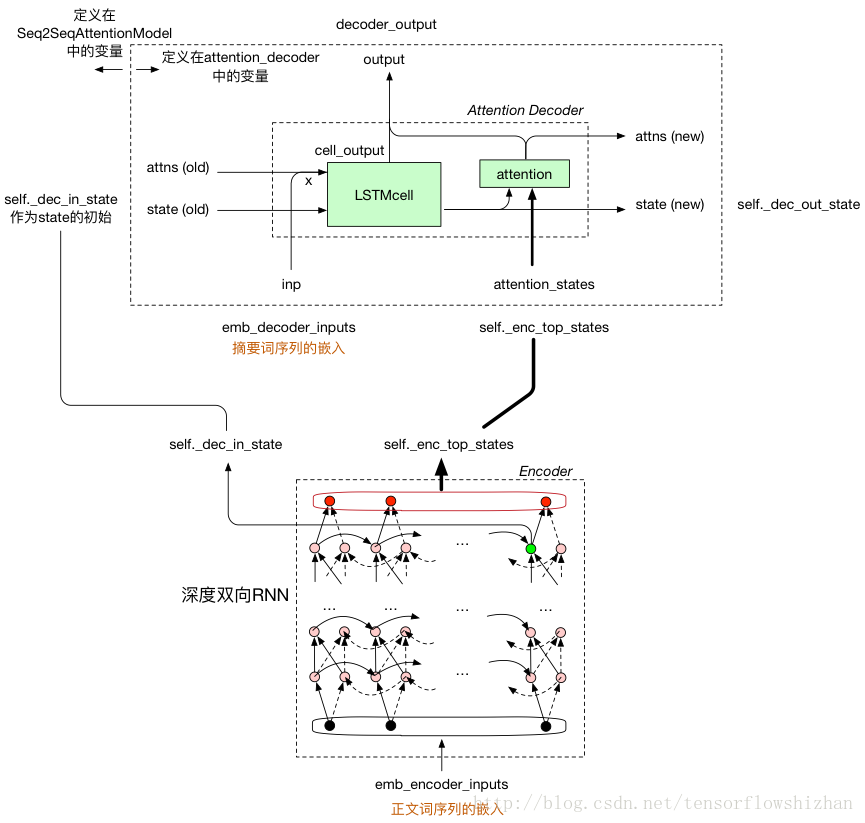
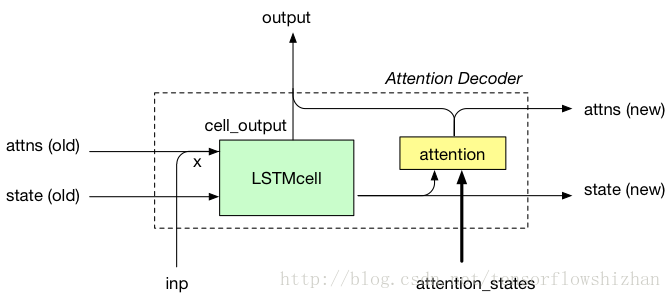
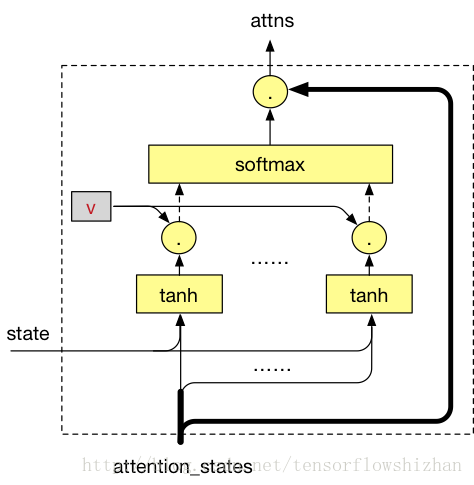
1. π[0]是可以用规定好的起始符号<s>来初始化. 在训练和生成摘要时, 窗口Q和C沿着文本滑动如果超出范围, 用起始符号<s>做padding.
2. 如果模型是abstraction-based, 输出y的备选集合是整个字典, 如果希望摘要的单词全部从原文中抽取, 那么词典由输入正文x的所有单词构成.
3. 我们会设定一个最大输出长度N, 算法会进行N轮迭代.   
   1. 现已有K个假设, 每一个假设都对应一条路径; 对每一个假设, 我们从字典S(有V个单词)中选出K个单词作为备选.
   2. 在字典中寻找, 搜索其他单词, 如果计算的到的state值比当前集合中的任意一个大, 就把它保留下来.
   3. 当每一个假设都遍历完整个字典S, 就会产生K×K条路径, 我们在这些路径中选择概率最大的K个路径作为下一次迭代的基础.(每一条路径都保留了之前i−1个节点对应的单词)
4. 当N次迭代进行完后, 我们只剩下了K条路径, 最后在从这其中选出1条概率最大的即可.
5. 路径所经历的所有节点即为摘要的单词. 如果这中间遇到了停止符<e>, 摘要就是从<s>到<e>, 如果没有<e>出现, 摘要的最大长度就是N.

Beam Search的运算复杂度从O(NVC)变成了O(KNV), 因为V>>N和K, 加速效果非常显著. 束搜索依据已经计算好的路径以及当前的V个备选值, 计算出最优的K的值. 最新的K个最优值都保留着相应路径上之前的所有的节点.

3 TensorFlow程序实战

NAM模型的程序最早是由facebook开源的torch版本的程序. 最近谷歌开源了TensorFlow版本的摘要生成程序textsum, Github上的项目. textsum的核心模型就是基于注意力的seq2seq(sequence-to-sequence)模型, textsum使用了LSTM和深度双向RNN.   
Github上的textsum首页给出了此项目在Bazel环境下的运行方式. 如果你不想通过Bazel运行, 你可以直接在seq2seq\_attention.py中设定运行参数. 设定完参数后, 直接运行**Python** seq2seq\_attention.py即可. 参数设定如下图所示:   
   
除了上述项目运行时所需的必要参数, 模型参数也在seq2seq\_attention.py中设定, 如下图所示, 包括学习率, 最小学习率(学习率会衰减但不会低于最小学习率), batch size, train模式encoder的RNN层数, 输入正文词汇数上限, 输出摘要词汇数上限, 最小长度限制, 隐层节点数, word embedding维度, 梯度截取比例, 每一个batch随机分类采样的数量.   
   
**Git**项目textsum给的toy数据集太小, vocab也几乎不可用(一些常见的单词都没有覆盖到). 如果希望获得好的效果, 需要自己整理可用的数据集.   
**主要文件说明**:   
- seq2seq\_attention.py: 主程序, 选择程序的运行模式, 设定参数, 建立模型, 启动tensorflow   
- seq2seq\_attention\_model.py: 建立attention-based seq2seq model, 包括算法的encoder, decoder和attention模块, 都在Seq2SeqAttentionModel中完成.   
- seq2seq\_attention\_decode.py: 读取数据, 调用beam\_search解码   
beam\_search.py: beam search算法的核心程序

textsum程序解析

Google开源的textsum项目的具体算法是基于Hinton 2014年的Grammar as a Foreign Language这篇论文, 下面给出textsum工程中attention-based seq2seq模型的整体结构图, 图中所使用的名字与程序中的变量名一致, Seq2SeqAttentionModel是一个类, 定义在seq2seq\_attention\_model.py中; attention\_decoder是一个函数, 定义在/tensorflow/contrib/legacy\_seq2seq/python/ops/seq2seq.py中.   
为了方便理解, 简单解释一下图中出现的符号,   
   
第一个符号表示从x1,x2到y的线性变换, 红色变量是训练过程要学习出来的.   
   
attention机制比较复杂也比较重要, 我们对这部分细化一下来看. attention decoder结构图如下:   
   
下图是对attention模块的细化:   
   
符号说明:   


为什么attention这个模块会起到效果呢? 因为attention模块会根据decoder当前时刻的LSTM单元的状态, 来调整对attention\_states(encoder输出)的注意力. Attention\_states不同位置获得的关注不一样. 这样我们就更大程度地, 关注了原文中, 对当前输出更为有用的信息, 输出结果也就更准确了. Attention模块输出结果和decoder模块原本的输出联合起来, 得到最终的输出结果.